Novel MC/DC Coverage Test Sets Generation Algorithm, and MC/DC Design Fault Detection Strength Insights

Mohamed A. Salem
Computer Science Department, Faculty of Engineering
University of Bristol, UK
Email:csmams@my.bristol.ac.uk

Kerstin I. Eder
Computer Science Department, Faculty of Engineering
University of Bristol, UK
Email:kerstin.eder@bristol.ac.uk

***Abstract*—This paper introduces Modified Condition/Decision Coverage (MC/DC), novel MC/DC coverage test sets generation algorithm named OBSRV, and MC/DC design fault detection strength. The paper presents an overview about MC/DC in terms of the MC/DC definition, MC/DC types, and the conventional MC/DC approaches. It introduces a novel algorithm, called OBSRV, for MC/DC coverage test sets generation. OBSRVresolves MC/DC controllability and observability by using principles found in the *D*-algorithm that is the foundation for state-of-the-art ATPG. It thereby leverages hardware test principles to advance MC/DC for software, and hardware structural coverage. The paper presents an investigation of the introduced OBSRValgorithm scalability, and complexity to prove its suitability for practical designs. The paper investigates MC/DC functional design faults detection strength, and analyzes empirical results conducted on main design fault classes in microprocessors.**

I. Introduction and Motivation

The main objective of the verification process is to identify design faults that have been introduced during the transformation of design specification into implementation source code, *ie* RTL. Coverage is a metric of verification progress versus the verification plan objectives as functional coverage, and versus the implementation source code as structural coverage.

Requirements coverage analysis determines the design implementation compliance with the functional requirements, and establishes traceability between the requirements, and the functional tests. Structural coverage analysis aims to establish how well the requirements-based test suite has exercised the design implementation source code structure, and determines code coverage gaps. In practice, especially for safety-critical applications, requirements coverage analysis is combined with structural coverage analysis to ensure high level of confidence that the design implementation has no unintended functions.

MC/DC coverage is a structural coverage metric originally used for the certification of critical software for avionics applications as required by RTCA/DO-178B [1]. MC/DC is a form of expression coverage demonstrating that the implementation has controllable and observable control flow logic [2]. MC/DC introduces cost-effective coverage solution in terms of the linear growth of the size of coverage test set to the exponential growth that requires 2^N tests, where N is the number of conditions in a decision statement.

MC/DC test generation requirements are complex due to the fact that there are mutual dependencies between tests. Recently, model checkers have been utilized to construct MC/DC coverage test sets from counter examples obtained by presenting a model checker with properties that negate MC/DC test requirements [3]. This paper takes a different angle to MC/DC coverage test sets generation, and the novel algorithm developed, called OBSRV, is purely based on fulfillment of the controllability and observability requirements of MC/DC [4]. The implementation of the OBSRV algorithm takes advantage of the path-oriented test generation algorithms such as the D-algorithm [5], [6], [7], and the PODEM algorithm [8] which are well understood in the context of Automatic Test Pattern Generation (ATPG), and also utilizes a greedy algorithm for test case prioritising to target MC/DC coverage [9].

II. Paper Outline

This paper is organized as follows: Section III introduces the paper definitions and notations, Section IV gives an overview of MC/DC coverage metrics in terms of MC/DC coverage definition, independence pairs, MC/DC coverage types, and MC/DC conventional approaches. Section V introduces the OBSRVusage model, *D-algorithm* overview, OBSRVprocedures, a simple example elaborating the OBSRVusage model, and ends with an investigation of the OBSRValgorithm scalability, and complexity. Section VI experimentally evaluates the MC/DC functional design fault detection strength versus common design fault classes in microprocessors. Finally, Section VII summarizes and concludes with an outlook on future research.

III. Definitions and Notations

The following notational conventions are used throughout the paper: boolean operators are denoted by **and**, **or**, **xor**, **not**. Boolean conditions and decisions are denoted by bold capitals such as **A, B, C**, etc. Truth values are written as either *true* and *false*, or *T* and *F*. A test for a boolean function with n inputs is denoted by $V = (V_1 V_2 \cdots V_n)$, where V_i is *T* or *F*. Test vectors can be represented by their equivalent decimal value, *ie* the test $V = (FTFT)$ is referred to as test 5.

IV. Modified Condition Decision Coverage Overview

A. MC/DC Definition

The Modified Condition/Decision Coverage criterion was developed to achieve many of the benefits of Multiple Condition/Decision testing while retaining a linear growth in required test cases [10]. The formal definition of MC/DC requires that: *every point of entry and exit in the design description or software program has been invoked at least once, every condition, and decision has taken all possible values at least once, and each condition in a decision has been shown to independently affect the decision outcome. The independence requirement ensures that the effect of each condition is tested relative to the other conditions* [2].

B. MC/DC Independence Pairs

A condition independently affects a decision outcome if that condition *alone* can determine the value of the decision outcome. Two test cases that show the independent effect of a condition within a decision are referred to as an independence pair [11]. For example, test vectors *(FT,TT)*, and *(FF,TF)* each form an independence pair for a 2-input **and** gate, and a 2-input **or** gate respectively due to the independent effect of the changing condition on the output value.

C. Unique-Cause and Masking MC/DC Approaches

For Unique-Cause MC/DC, a condition is shown to independently affect a decision outcome by varying just that condition while holding all other possible conditions fixed. For Masking MC/DC, a condition is shown to independently affect a decision outcome by applying principles of boolean logic to assure that no other condition influences the outcome, though more than one condition in the decision may change value. The mathematical representation of the unique-cause MC/DC independence pair implies that the boolean difference function $\delta\mathbf{F}(\mathbf{c})/\delta\mathbf{c_i} = \mathbf{F}(\mathbf{c_1}, ..., \mathbf{c_i}, ...\mathbf{c_n}) \oplus \mathbf{F}(\mathbf{c_1}, ..., \neg\mathbf{c_i}, ...\mathbf{c_n})$ is always *true* such that $F(c)$ is an expression with *n* inputs [12].

For Masking MC/DC approach the equation will remain valid taken into account that condition c_j from j = (1,..,n) and $j \neq i$ might change its values but their influence are masked by the logic under investigation. For example, a *true* input for a 2-input **or** gate will mask the other input, and similarly a *false* input will have a dominant effect on a 2-input **and** gate and mask the other input. However, this requires analysis of the decision statement logic structure [10]. NASA has a special approach for evaluation of the MC/DC coverage fulfillment for software code. It uses the requirements-based testing to check for MC/DC as per software certification requirements in [1], and elaborated in [2].

V. MC/DC Coverage Test Sets Generation Algorithm: OBSRV

This section presents a novel MC/DC coverage test set generation algorithm, called OBSRV. The main goal of MC/DC coverage is to show the independent effect of each input condition on the value of the decision by controllability and observability aspects as shown in [4], and [13]. Figure 1 shows a complex gate structure comprised of **and**, **or**, **xor**, and

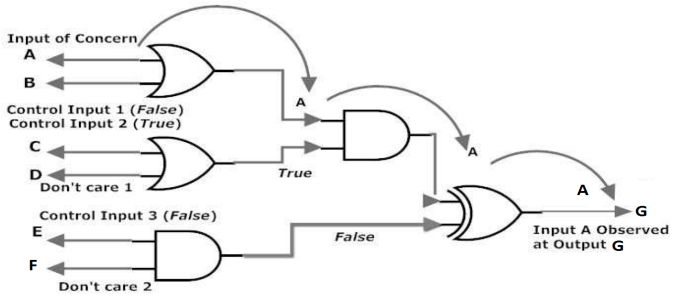

Fig. 1: Controllability and Observability

shows how the value of primary input **A** can be propagated to be observable at the final output **G**. The foundation of the OBSRValgorithm is based on the path-oriented test generation algorithms that have been used in the hardware fault detection test generation such as *D-algorithm* I [5], *D-algorithm II* [6], or PODEM [8] algorithm. Subsection V-A provides an overview about the OBSRVusage model, and Subsection V-B provides an overview about the *D-algorithm* which is the basis of OBSRV. OBSRVoperates as per the procedure flow shown in Figure 2, and elaborated in Subsections V-C, V-D, V-E respectively.

A. OBSRV Usage Model

OBSRVmain objective is to generate MC/DC coverage test sets for a single decision statement of control flow logic within the design code structure. The control flow logic analysis based on controllability, and observability aspects is the foundation of OBSRV. This analysis combined with unique, and masking MC/DC concepts enable the determination of all possible test vectors that fulfill the observability requirements for the decision statement input conditions. This process guarantees that OBSRVwill generate the maximal number of MC/DC coverage test sets, and the minimal number of tests per particular generated MC/DC coverage test sets. The maximal MC/DC generated test sets will enable an efficient MC/DC coverage closure by the functional regression suite, and determination of the MC/DC coverage gaps. The minimal number of tests per MC/DC test set provides an optimal MC/DC coverage closure. Hence, OBSRVusage model has a potential added value to the efficiency, and the optimization of the MC/DC coverage closure process.

B. D-Algorithm Overview

The *D-algorithm* is developed to generate tests for fault detection for digital circuits, and this subsection will briefly provide an overview with details found in [5], [6], and [7]. *D* represents a generic truth value, and the *D-cube* is a collapsed truth table version of a particular logic operator using *D*. The *D-algorithm* starts with representation of the expression by gate level schematic showing the primary input/output, and internal nodes. The next step will be to choose a primary output, determine a path to this output, and propagate the fault to the output using the propagation *D-cubes* (PDC) of the logic gates of this path. This path is called the *D-chain*, and the propagation D-cubes *(PDC)* will enable the propagation of the fault in terms of the logic gate types, and input conditions values. The *singular cover* is defined as the truth table structure for all the gates that construct the logic circuit. The final step

is called the *consistency* operation, and this step determines consistent values of the unassigned nodes within the logic structure. The consistency is relative to being consistent with the implications of the *singular cover*.

C. Conditions Observability Computation

The graph representation of the decision statement as netlist structure is the entry point to the algorithm. This OBSRVphase uses the *D-algorithm* to define the *D-chain(s)* of each primary input *D* value propagation. Each primary input *D-chain* consists of a listing of all those gates whose primitive *D-cube* intersections do not lead to inconsistent line assignments [6]. These consistent *D-intersections* will lead to defining the assignments of the nodes at each gate of the *D-chain* that will drive the selected primary input ***D*** value propagation to the decision output node. The assignment through a particular gate to propagate the ***D*** value represents an imposed implication of other nodes assignment across the logic. For example, a *D* propagation through an **and** gate will require a *true* input, and a *false* input through an **or** gate. This phase is completed by the consistency operation which is the completion of the resultant *D-cube* by *D-intersection* with the singular cover [5].

Masking MC/DC enables possibilities during the consistency operation. For example, if an output of an **and** gate is *false*, there is three possibilities of the input *D-coordinates* as *(FF,FT,TF)*. Bimodal decision is defined as an expression with *D-chains* that can observe $\bar{D}$ which is the negated value of *D* as well as the *D* absolute value. The bimodal decisions *D-chains* must contain at least one negating or inverting logic operator.

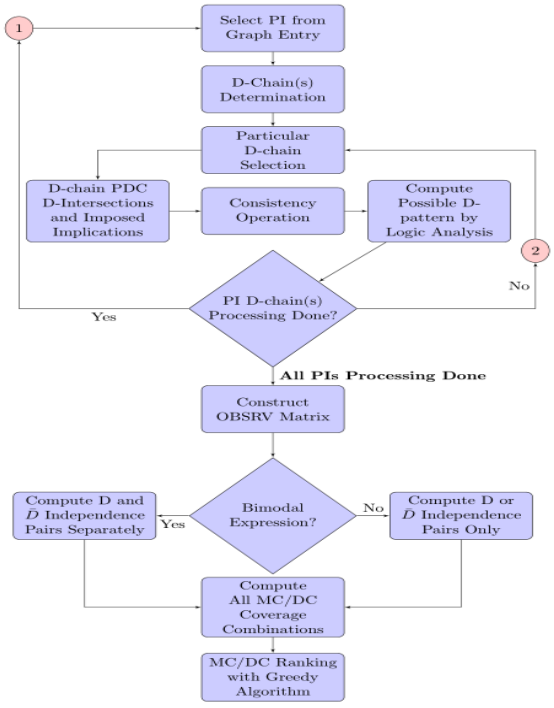

Fig. 2: OBSRV Algorithm Flowchart

D. Observability Matrix, and Independence Pairs Inference

The second phase of OBSRVis the construction of the observability matrix. This matrix states all the possible input patterns (*D* test vectors) that result from each primary input observability through all possible *D-chains*, and the corresponding observed value at the output whether *D* or $\bar{D}$. OBSRVwill check from the matrix about the possibility of bimodal exhibited processing for a particular primary input if it can be observed as negated and non-negated (*D* and $\bar{D}$) through different *D-chains*. In case of bimodal processing of a primary input the computation of the independence pairs for the non-negated *D-chains* must be done separately from the negated *D-chains*. OBSRVwill compute the possible independence pairs for an input in two steps; the first will be unique-cause by replacing *D* by *true*, and *false* for a certain observability pattern. The masking independence pairs are computed by combining a pattern for *true* value observation and another pattern for *false* value observation in the matrix such that both patterns were leading to the same *D* observation whether negated or non-negated.

E. MC/DC Test Sets Combinations, and Prioritization

OBSRVwill generate all possible MC/DC coverage test sets by combining one independence pair for each condition (primary input) in the logic expression. The combined independence pairs test vectors will formulate the required MC/DC coverage test set. The ranking of the coverage test sets will be accomplished using a greedy algorithm to rank the coverage set in an ascending order of their test vectors count [9], [14].

F. OBSRV Example

This section presents a manual example of MC/DC coverage test sets generation using the OBSRValgorithm. The example used is the same used in [5] as shown in Figure 3. The primary output function is **G**, and $\mathbf{G} = (((\mathbf{A}\ \textbf{and}\ \mathbf{B})\ \textbf{or}\ \mathbf{C})\ \textbf{xor}\ \mathbf{A})\ \textbf{nand}\ (\textbf{not}\ ((\mathbf{A}\ \textbf{and}\ \mathbf{B})\ \textbf{or}\ \mathbf{C})\ \textbf{or}\ \mathbf{E})$ The possible *D-chains* for each primary input will be determined by the *D-cube* intersections, and the patterns that will lead to observability at the output will be completed by the imposed implications, and the consistency operation as explained in Subsection V-C. The observability matrix is constructed as shown in Table I. The primary input **C** exhibits bimodal expression processing as the observed primary output could be non-negated as *D*, or negated as $\bar{D}$.

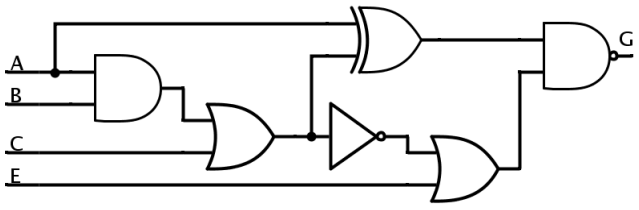

Fig. 3: OBSRV Example

The full list of possible independence pairs from unique-cause, and masking MC/DC will be computed by replacing *D* by *true*, and *false* values, and taking into account the bimodal combination of the **C** primary input. **C** independence pairs should be computed based on the patterns that results in D or $\bar{D}$ values at the primary output separately as explained in Subsection V-D. The full list of MC/DC coverage sets will be 512. The greedy algorithm will be used to rank the coverage sets in an ascending order in terms of the number of tests per each

coverage set as explained in V-E. For example, the coverage set (2,3,8,11,12) has 5 tests, which is one of the optimal MC/DC coverage test sets for this example. An elaborated empirical MC/DC study using the OBSRValgorithm, and an associated set of novel MC/DC insights have been deduced [15].

	A	B	C	E	G
A	D	0	1	1	D
	D	1	1	1	D
B	1	D	0	0	D
	1	D	0	1	D
C	1	0	D	0	D
	0	0	D	1	$\bar{D}$
	1	0	D	1	D
	0	1	D	1	$\bar{D}$
E	0	0	1	D	$\bar{D}$
	0	1	1	D	$\bar{D}$

TABLE I: Observability Matrix

A	(3,11), (3,15), (7,15), (7,11)
B	(8,12), (8,13), (9,13), (9,12)
C	(8,10), (8,11), (9,11), (9,10), (1,3), (1,7), (5,7), (5,3)
E	(2,3), (2,7), (6,7), (6,3)

TABLE II: Independence Pairs

G. OBSRV Scalability and Complexity Analysis

This subsection studies the scalability analysis of the OBSRV algorithm, the various types of introduced scalability or complexities, the investigation of the advantages, and limitations. It provides statistical empirical results for the computational run-time, memory usage, and debugging information changes versus the scalability of a particular design under test.

1) Horizontal and Vertical Design Scalability: The horizontal scalability defines as the growth of the number of conditions in a particular decision statement. The vertical scalability defines as the growth of the number of decision statements in a particular design under test. The research work has adopted the path oriented test generation while the development of the OBSRV algorithm. The complexity of the path sensitization methods have been investigated by Bushnell, and Agrawal [16]. The analysis has shown that the complexity is exponential in circuit size. The PODEM algorithm has exponential complexity, but it has an advantage of several orders faster than the D-algorithm. The D-algorithm considers propagation of an internal fault node within the logic structure of the circuit. The PODEM algorithm takes into account the propagation of the primary circuit inputs only. OBSRV algorithm studies only the propagation of a primary input, such that the controllability and observability requirements can be achieved for each condition or primary input of the circuit. Hence, it takes advantage of the faster performance feature. Table III states the statistics of the number of conditions, and the corresponding number of expressions as found in avionics systems which are intensively complex. The exponential algorithm complexity with respect to its asymptotic behavior versus the expression size will not have a considerable practical impact as shown by the data. The number of conditions that can influence the complexity does not occur in practical Boolean expressions typically found in avionics systems as shown in this example data [2].

2) Operand Chain Effect: The operand chain effect defines as the number of logic operands/operators per a particular decision statement. The extreme case of the chain effect is represented by chip end-to-end. This is the cascaded logic

n In	1	2	3	4	5	6-10	11-15	16-20	21-35	36-76
Exp.	16491	2262	685	391	131	219	35	36	4	2

TABLE III: Avionics complexity: Expressions versus inputs

levels from the chip primary input to the chip primary output. OBSRV generates MC/DC coverage test sets per decision statement, and checks for MC/DC fulfillment versus the stimulus generation from the requirements-based tests. It is not applicable to check a complete design from end-to-end. The growth of logic levels per a single decision statement is proportional to the growth of the conditions, and decisions as in horizontal, and vertical scalability case stated in V-G1.

3) D-drive Options Scalability: The D-drive options scalability defines as the growth of the D-propagation paths per a particular condition of the decision statement. The alternative logic paths or D-drives or D-fronts or D-propagation paths are mainly due to the contribution of the same unique condition or primary input to the various operands or the gates at the different logic levels. This results in the optional different paths to propagate that specific unique input to the output node. This type of scalability does not result in an exponential complexity growth. This results in multiple iterations of the algorithm procedures to address all the Primitive D-cube intersections for each primary input path until they are all exhausted or completed. This scalability will add advantages as the number of generated observability objectives tests will increase. It will enable scalability of the generated MC/DC coverage test sets.

4) Experimental Scalability Investigation: This experiment has been conducted in order to study the scalability aspects of the OBSRV algorithm under its current implementation. The results indicate no considerable negative impact on the scalability of the OBSRV algorithm implementation under the practical spectrum of the number of conditions, and decisions in a specific design under test. The experiment has formulated an incremental growth of the complexity by adding complex concurrent decision statements from the unimodal type, and the bimodal type. The scalability magnitude of order was measured by the quantitative absolute number of logic operands under processing by the MC/DC algorithm implementation. The machine used has quad-core with 16 GB memory size under Linux. The unimodal expression has incrementally grown to a scalability order of magnitude to X1000 times the original design. The elapsed time from the transcript evaluates to zero seconds which is in magnitude of fraction of seconds. The memory, and the debugging information complexity has been growing to 500MB order. The experiment has been conducted using bimodal decision statements. The incremental complexity has been growing until reaching X4000 absolute value of complexity growth. The measured elapsed time is zero, which implies an actual execution run time in the order of fractions of seconds. The memory usage has no limited impact or consumption of the memory resources. The debugging information size from the transcript, and log files grow significantly from 14KB to 1300KB. The experimental investigation shows that the execution run times under a practical number of conditions are manageable. It will not reach the limits that will represent an obstacle to complete the computation of the algorithm. The main impact is represented by the growth of the size of the debugging information represented by the coverage reports, the transcripts, and the debugging log files.

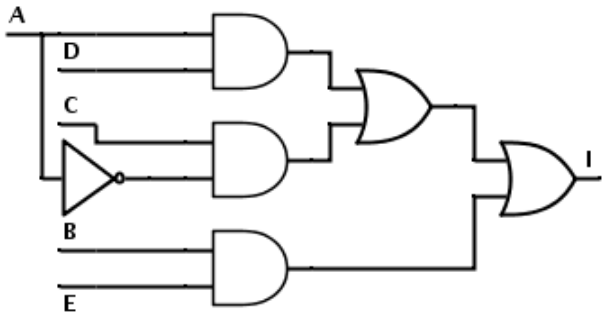

Fig. 4: Design Fault Case Study Expression

VI. MC/DC Design Fault Detection Strength

This section investigates the MC/DC design fault detection strength in terms of the main design fault classes in microprocessors as stated in [17], and [18]. The experimental investigation conducted on a diverse logic expressions from industrial designs with unimodal, and bimodal expression types, using main logic operators, and variant number of input conditions. The concluded results are elaborated using the case study expression **I = (A and D) or (not A and C) or (B and E)** of [17], and shown in Figure 4. The MC/DC coverage test sets have been generated using OBSRVprocedures stated in Section V, and sample of the generated test sets are (1,2,3,4,8,16,24),(1,2,4,8,16,19,25). The experiment procedure requires each design fault type to be applied to the expression, and the MC/DC coverage results and the functional output of the expression will be evaluated for all fault types as shown in the following Subsections VI-A, VI-B, VI-C, VI-D VI-E, VI-F, VI-G, VI-H, VI-I, VI-J. Questa 10.0 FEC is used for checking MC/DC coverage fulfillment.

A. Expression Negation Fault: ENF

I = not [(A and D) or (not A and C) or (B and E)] represents the expression with ENF design fault applied. MC/DC full coverage has been achieved while exercising the expression using the MC/DC test set of the original expression. The logic analysis shows that the logic structure has an additional inverter gate introduced before the primary output node. The expression with full ENF is still controllable, and observable due to the transparent path to the output node. The only difference is the inversion of the output value. MC/DC targets controllability and observability of the input conditions by having an independent effect on the decision, and still this will be applicable. The fault is easily detected as the full ENF type enable transparent propagation of the fault to the output, which will be detected by comparison of the functional output truth value of the original expression which will be negated.

B. Subexpression Negation Fault

I = not ((A and D) or (not A and C)) or (B and E) represents the expression with partial ENF applied to the first, and second term. The MC/DC coverage test set does not fulfill MC/DC in this fault. The experiment FEC results indicated MC/DC coverage gaps for B_1, and E_1; the *true* values of **B**, and **E** respectively. The logic analysis indicates a logical structure change which implies a change in the controllability, and observability requirements of the decision statement. The observation of B_1, and E_1 will be masked by a *true* value on the other input to the output **or** gate in Figure 4. The MC/DC test set for the original expression must enable observability of B_1, and E_1 by *false* input to the **or** gate, and this was negated by introduction of an inverter after the **or** at the second logic level. The MC/DC test set detects the introduced fault by presenting MC/DC coverage gaps in the FEC results.

C. Term Negation Fault: TNF

I = not (A and D) or (not A and C) or (B and E) represents the expression with TNF fault applied to the first term. The experiment FEC results indicated MC/DC coverage gaps for B_1, E_0, E_1. MC/DC test set enables detection of the fault, and the logic analysis indicates a change of the logic structure of the expression that impacts the controllability, and observability requirements for input conditions.

D. Term Omission Fault: TOF

I= (not A and C) or (B and E) represents the expression with TOF fault applied to the first term. The experiment FEC results indicated MC/DC coverage gaps for A_1. The logic analysis indicates that the omission of the first term resulted in a change of the logic structure such that the expression changed to be a unimodal expression. The original expression is a bimodal expression as it has an inverting and non-inverting path to the output node for the input condition **A**. The omission of the term (**A and D**) removed the non-inverting observability path of the input **A**. MC/DC test set detects the TOF fault by showing the FEC coverage gaps in the results.

E. Literal Omission Fault: LOF

I = D or (not A and C) or (B and E) represents the expression with LOF fault applied to first literal **A** of the first term (**A and D**) by omission of **A**. The MC/DC test set shows coverage gap in FEC for A_1. The logical analysis shows an introduced logic structure change as the **D** input condition will bypass an **and** gate and directly pass to the **or** gate. This will change controllability, and observability requirements of **A** that was shown by the MC/DC FEC coverage gap.

F. Literal Insertion Fault: LIF

I = (A and D and B) or (not A and C) or (B and E) represents the expression with LIF applied to the first term by adding **B** to the first **and** gate. The MC/DC test set detects the introduced fault by showing FEC coverage gaps A_1, D_0, D_1. The logic analysis implies a logic structure change that impacts the controllability, and observability requirements for MC/DC.

G. Literal Negation Fault: LNF

I = (not A and D) or (not A and C) or (B and E) represents the expression with LNF applied to the literal **A** of the first term. The MC/DC test set achieves FEC coverage for the expression. The logic analysis shows that there is no substantial change to the expression logic structure, as adding an inverter gate after the primary input **A** does not

impact the controllability, and observability requirements for MC/DC. The functional coverage suite detects a change in the output truth values of the expression compared to the original expression which enables a detection of an introduced fault.

H. Literal Reference Fault: LRF

I = (B and D) or (not A and C) or (B and E) represents the expression with LRF applied to the literal **A** of the first term by replacing it by the input condition **B**. MC/DC test set detects the fault by showing FEC coverage gaps for A_1, and D_1. Logic analysis shows that a logic structure change is introduced by LRF due to a change of one primary input by another, which impacts the controllability, and observability requirements for the original expression MC/DC fulfillment.

I. Disjunctive Operator Reference Fault: ORF(+)

I = (A and D) and (not A and C) or (B and E) represents the expression with ORF(+) fault by changing the **or** gate to an **and** gate at the second logic level. MC/DC test set detects the fault by showing coverage gaps for primary inputs **A**, **C**, and **D**. The logic analysis shows a major change in the logic structure of the expression as it will be simplified to only an **and** gate between **B**, and **E**. MC/DC shows that **A**, **C**, and **D** are redundant inputs by showing these inputs as *Not Testable* in the FEC report. The logic structure change has an impact on the controllability, and observability requirements of MC/DC, which leads to the fault detection.

J. Conjunctive Operator Reference Fault: ORF(.)

I = (A or D) or (not A and C) or (B and E) represents the expression with ORF(.) fault by replacing the **and** gate by **or** gate for inputs **A**, and **D** at the first logic level. MC/DC test set detects the fault by showing coverage gaps for primary inputs **B**, and **E**. The MC/DC FEC coverage results shows gaps for B_1, and E_1 The logic analysis shows a change in the logic structure of the expression which has an impact on the controllability, and observability requirements for MC/DC, which leads to the fault detection.

VII. Summary and Conclusion

This paper promotes the adoption of MC/DC in hardware verification as a structural coverage metrics. The hardware verification requirement for an efficient structural coverage metrics, the success of MC/DC in critical software applications, the description of hardware using HDL and the MC/DC computation dependency on logic analysis have been potential motivational factors for this research. The paper has provided an overview about the MC/DC types as unique-cause and masking MC/DC with an explanation of the main concepts of MC/DC represented by the controllability and the observability aspects. The research work has produced OBSRV, which is a novel algorithm for MC/DC coverage test sets generation. The main steps in the procedural computation of OBSRVhave been outlined with detailed example. The example presents a manual implementation of OBSRVon selected logic function to enable clear understanding of the algorithm technique. OBSRVis based on path-oriented test generation algorithms as *D-algorithm* I, II which established the fundamental work of 48 years of state-of-the-art in the application of hardware fault detection test generation. OBSRVrepresents a potential bridge to portability of this technology advancement to be applied to MC/DC coverage for software, and hardware. The paper has presented an investigation of the OBSRValgorithm scalability, and complexity analysis. The horizontal, and vertical design scalability, operand chain effect, D-drive options scalability, and an experimental scalability investigation have been introduced. The paper has investigated the MC/DC functional design faults detection strength, and the empirical results conducted on the main design fault classes in microprocessors demonstrated sufficient MC/DC detection capability of the various introduced design faults. The paper research results will enable continuity of research in terms of MC/DC adoption as main metric for functional verification coverage closure.

References

[1] *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, Radio Technical Commission for Aeronautics (RTCA) Std., 1982.

[2] K. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson. (2001) A Practical Tutorial on Modified Condition/Decision Coverage.

[3] S. Rayadurgam and M. Heimdahl, "Generating mc/dc adequate test sequences through model checking," in *Software Engineering Workshop, 2003. Proceedings. 28th Annual NASA Goddard*, dec. 2003.

[4] M. Abramovici, A. Friedman, and M. Breuer, *Digital Systems Testing and Testable Design*. IEEE, 1999.

[5] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM Journal of Research and Development*, vol. 10, no. 4, pp. 278 –291, july 1966.

[6] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits," *IEEE Transactions on Electronics Computers*, vol. EC-16, no. 20, pp. 567 –580, oct. 1967.

[7] W. Bouricius, E. Hsieh, G. Putzolu, J. Roth, P. Schneider, and C. Tan, "Algorithms for detection of faults in logic circuits," *IEEE Transactions on Computers*, vol. C-20, no. 11, pp. 1258 – 1264, nov. 1971.

[8] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *Computers, IEEE Transactions on*, vol. C-30, no. 3, pp. 215 –222, march 1981.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[10] J. Chilenski and S. Miller, "Applicability of modified condition/decision coverage to software testing," *Software Engineering Journal*, vol. 9, no. 5, pp. 193 –200, sep 1994.

[11] C. A. S. Team, "Rationale for accepting masking mcdc in certification projects," Position Paper 6, Tech. Rep., August 2001.

[12] Z. Kohavi, *Switching and finite automata theory*, ser. McGraw-Hill computer science series. McGraw-Hill, 1978.

[13] S. Devadas, A. Ghosh, and K. Keutzer, "An observability-based code coverage metric for functional simulation," *Computer-Aided Design, International Conference on*, vol. 0, p. 418, 1996.

[14] J. Jones and M. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," in *Software Maintenance, 2001. Proceedings. IEEE International Conference on*, 2001, pp. 92 –101.

[15] M. A. Salem and K. I. Eder, "Modified Condition Decision Coverage: A Hardware Verification Perspective," in *IEEE 14th Microprocessor Test and Verification (MTV)*, December 2013, pp. 8–13.

[16] M. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-signal VLSI Circuits*. Springer, 2000, vol. 17.

[17] D. A. Mathaikutty, S. K. Shukla, S. V. Kodakara, D. Lilja, and A. Dingankar, "Design fault directed test generation for microprocessor validation," in *Proceedings of the conference on Design, automation and test in Europe*, ser. DATE '07. EDA Consortium, 2007, pp. 761–766.

[18] M. F. Lau and Y. T. Yu, "An extended fault class hierarchy for specification-based testing," *ACM Trans. Softw. Eng. Methodol.*, vol. 14, no. 3, pp. 247–276, Jul. 2005.